Business Case

Steven Lavenhar, Cigital, Inc. [vita[3]]

This article has been adapted with permission from "Software Quality at Top Speed" by Steve McConnell, 1996. For the original article, please see www.stevemcconnell.com[4].

2007-03-16 L3 / E, (L), M[5]

Substantial net improvements in application security have been obtained through the use of formal reviews of design and code. Improvements are made possible by a systematic design and code verification process. By using source code review results, a significant reduction in security flaws can be achieved. To perform this kind of analysis, it is necessary for a project to have defined, consistently executed processes for source code inspection, code rework, and retesting.

There are limited data available that discuss the ROI of reducing security flaws in source code. However, there are a number of studies available that have shown that significant cost benefits are realized through improvements in SDLC processes [Goldenson 2003[6]].

Software assurance in software development organizations is often underbudgeted and dismissed as a luxury. In an attempt to shorten their development schedules or decrease their costs, software project managers throughout the industry often reduce the time spent on software-assurance practices such as requirements analysis and design. In addition, they often try to compress the testing schedule or level of effort. Testing is very vulnerable to such reductions since it is a critical-path item in the development schedule. Skimping on software assurance is the worst decision an organization that wants to maximize development speed can make. In software development, higher quality (in the form of lower defect rates) and reduced development time go hand in hand. Figure 1 illustrates the relationship between defect rate and development time.

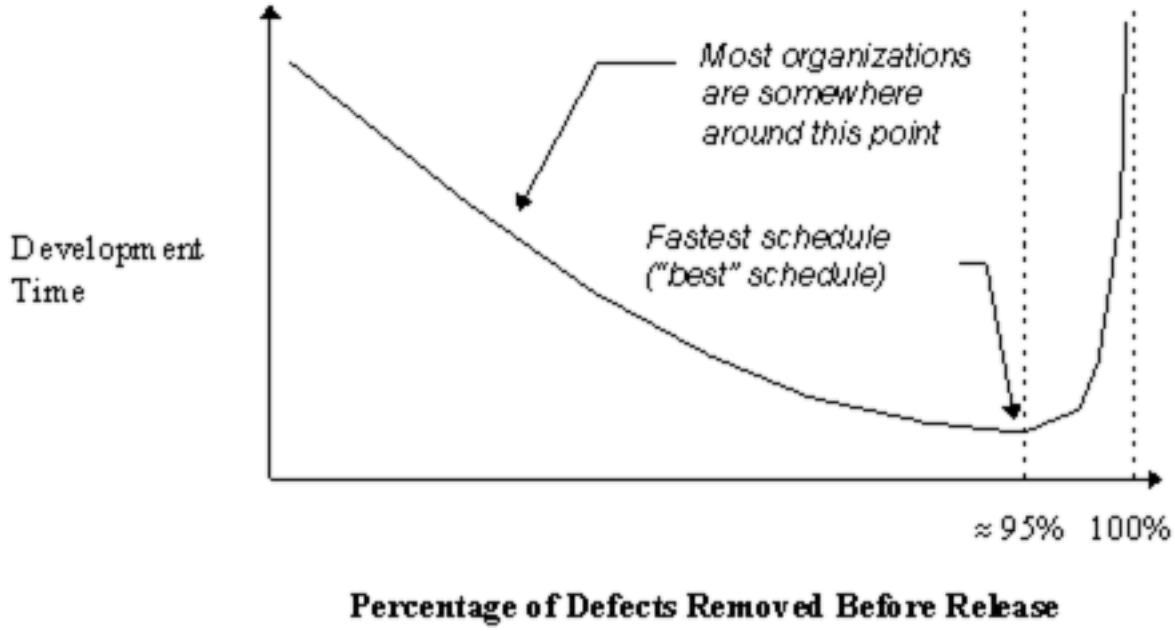

Figure 1. Relationship between defect rate and development time

As a rule, the projects that achieve the lowest defect rates also achieve the shortest schedules. Many organizations currently develop software with defect levels that give them longer schedules than necessary.

3. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/197-BSI.html (Lavenhar, Steven)
4. http://www.stevemcconnell.com/
6. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/code/213-BSI.html#dsy213-BSI_GG (Code Analysis - References)

In the 1970s, studies performed by IBM demonstrated that software products with the lowest defect counts also had the shortest development schedules [Jones 1991[7]]. Capers Jones [Jones 1994[8]] reported that poor quality was one of the most common reasons for schedule overruns after surveying over 4000 software projects. He also reported that poor quality was a significant factor in approximately 50 percent of all canceled projects. A Software Engineering Institute survey found that more than 60 percent of organizations assessed suffered from inadequate quality assurance [Kitson 1993[9]]. On the curve in Figure 1, the organizations that suffered from inadequate quality assurance are to the left of the 95-percent-defect-removal line.

The 95-percent-defect-removal line is significant because that level of pre-release defect removal appears to be the point at which projects achieve the shortest schedules, least effort, and highest levels of user satisfaction [Jones 1991[10]]. If more than 5 percent of defects are found after a product has been released, then the product is vulnerable to the problems associated with low quality, and the organization is taking longer to develop its software than necessary. Projects that are in a hurry are particularly vulnerable to shortchanging quality assurance at the individual developer level. Any developer who has been pushed to send out a deliverable or ship a product quickly knows how much pressure there can be to cut corners because "we're only three weeks from the deadline." Up to four times the normal number of defects are reported for released software products that were developed under excessive schedule pressure. Projects that are in schedule trouble often become obsessed with working harder rather than working smarter. Attention to quality is seen as a luxury. The result is that projects often work dumber, which gets them into even deeper schedule trouble.

One aspect of quality assurance that is particularly important to rapid development is the existence of error-prone modules, which are modules that are responsible for a disproportionate number of defects. Barry Boehm reported that 20 percent of the modules in a program are typically responsible for 80 percent of the errors. On its IMS project, IBM found that 57 percent of the errors clumped into 7 percent of the modules. Modules with such high defect rates are more expensive and time consuming to deliver than less error-prone modules. Normal modules cost about $500 to $1000 per function point to develop. Error-prone modules cost about $2000 to $4000 per function point to develop. Error-prone modules tend to be more complex than other modules in the system, less structured, and unusually large. They often are developed under excessive schedule pressure and are not fully tested. If development speed is important, then identification and redesign of error-prone modules should be a high priority. If a module is poorly structured, excessively complex, or excessively long, redesigning the module and reimplementing it from the ground up will shorten the schedule and improve the quality of the product at the same time.

If an organization can prevent defects or detect and remove them early, it can realize a significant cost and schedule benefit. Studies have found that reworking defective requirements, design, and code typically consumes 40 to 50 percent of the total cost of software development [Jones 1986[11]]. As a rule of thumb, every hour an organization spends on defect prevention will reduce repair time from three to ten hours. In the worst case, reworking a software requirements problem once the software is in operation typically costs 50 to 200 times what it would take to rework the problem in the requirements stage [Boehm 1988[12]]. It is easy to understand why. A one-sentence requirement can expand into five pages of design diagrams, then into 500 lines of code, 15 pages of user documentation, and a few dozen test cases. It is cheaper to correct an error in that one-sentence requirement at the time requirements are developed than it is after design, code, user documentation, and test cases have been written.

---

7. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/code/213-BSI.html#dsy213-BSI_jones91 (Code Analysis - References)
8. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/code/213-BSI.html#dsy213-BSI_jones94 (Code Analysis - References)
9. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/code/213-BSI.html#dsy213-BSI_KM (Code Analysis - References)
10. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/code/213-BSI.html#dsy213-BSI_jones91 (Code Analysis - References)
11. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/code/213-BSI.html#dsy213-BSI_jones86 (Code Analysis - References)
12. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/code/213-BSI.html#dsy213-BSI_BP (Code Analysis - References)

Figure 2 illustrates that the longer defects stay in a program, the more expensive they become to correct.

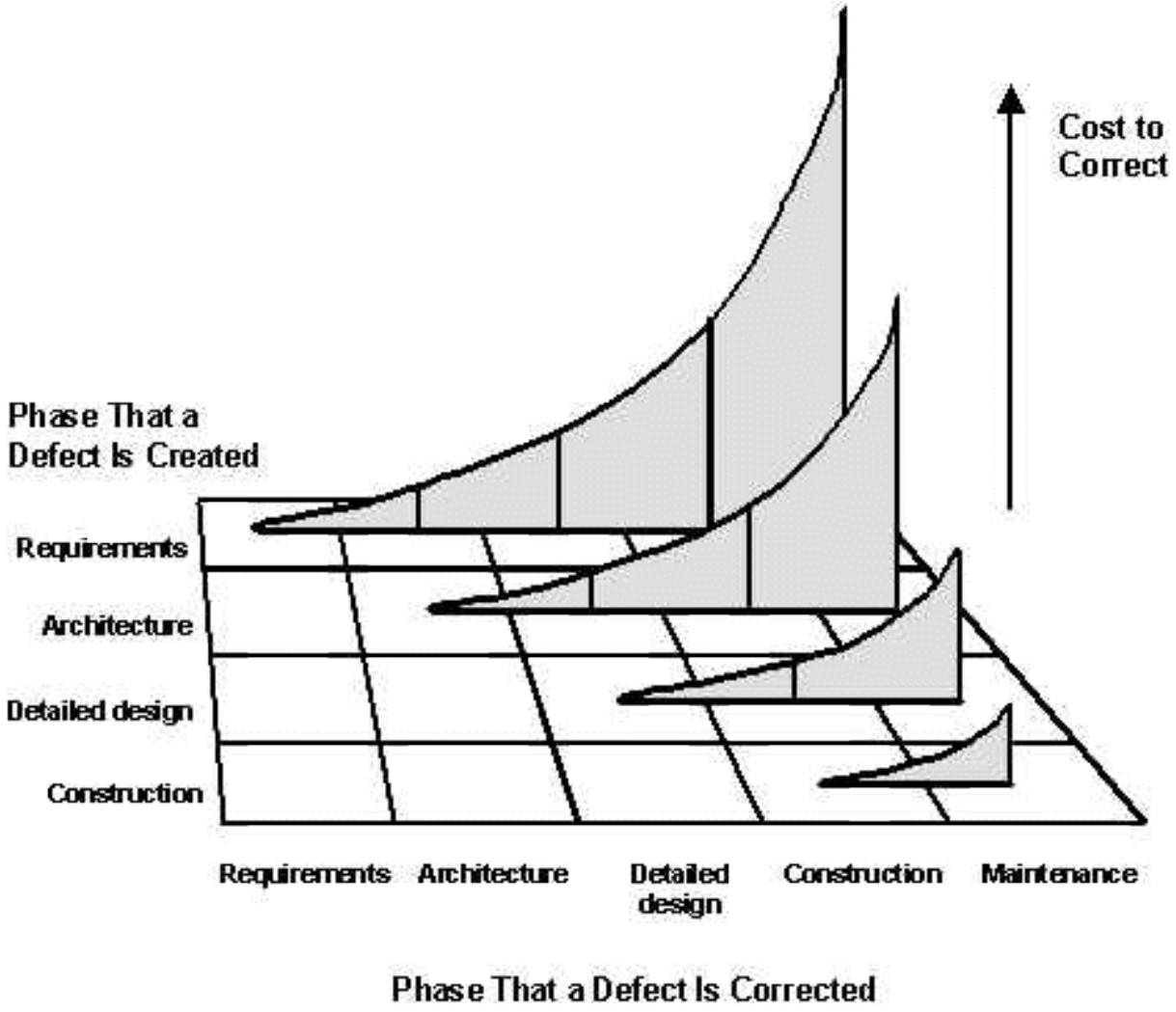

Figure 2. Cost of correcting defects by life-cycle phase

The savings potential from early defect detection is huge: about 60 percent of all defects usually exist by design time [Gilb 1988[13]]. A decision early in a project to not focus on defect detection amounts to a decision to postpone defect detection and correction until later in the project when they will be much more expensive and time consuming. That is not a rational decision when time and development dollars are at a premium.

Cigital, Inc. Copyright

13. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/code/213-BSI.html#dsy213-BSI_gilb (Code Analysis - References)